SAP Development Guideline

Filename:	sap_development_guidline.docx
Version:	2.1
Status:	Final Version
Last change:	06.06.2023

Abteilung TIS
Helge Hauffe
05531 702-95821
Helge.Hauffe@stiebel-eltron.de

1 Table of Content

2 Introduction

2.1 Responsibility

NAME	PROJECT ROLE	COMPANY
Andreas Henneke	Head of application development	Stiebel Eltron GmbH & Co KG

2.2 Change History

VERSION	DATE	AUTHOR	DESCRIPTION
1.0	15.03.2017	Helge Hauffe Daniel Weilbacher Heiko Weber	Initial Version
1.1	25.01.2019	Helge Hauffe	Added point 4.2.8
1.2	30.03.2021	Helge Hauffe	Added point 3.6
2.1	06.06.2023	Helge Hauffe Heiko Weber	Hana Version

2.3 Purpose

In such a complex system environment like SAP S/4 it is vital to keep track of all additional developments and modifications of standard coding. In SAP S/4 the program coding is written in the language ABAP/4.

When upgrading the system to a new release this information is needed to completely maintain or update custom system modifications or custom development. (Example: SAP changes from one software release to another the structure of a database table; all modifications and custom programs must be changed as well to maintain system integrity).

When system changes are necessary in the future the initial developers are usually not available any more. Therefore, certain rules must be adhered to that other developers can easily perform changes to the coding.

Such rules are:

- ✓ Naming conventions
- ✓ Clearly structured code
- ✓ Program documentation
- ✓ Modification standards
- ✓ Multi language support

3 Naming Conventions

3.1 Objective of Naming Conventions

Clarity and understandability of the source text is essential for easy evaluation, adaptation, and maintenance of programs:

- ✓ Clear and understandable code is more likely to be correct and reliable.
- ✓ Clarity fosters the thorough understanding of the software that is necessary for adaptation.
- ✓ Effective code adaptation is a prerequisite to code reuse and thus to reductions in system development cost.
- ✓ Clarity helps to keep maintenance costs low, because during the entire life cycle of the software, code has to be read and understood far more often than it is written.

This document defines naming conventions for developing ABAP programs in the customer area to make evaluation, adaptation, and maintenance of the programs easier.

Every object in ABAP has to be named in English.

3.2 Naming Conventions

The naming convention focuses on those objects, which are used during development very frequently and which are therefore relevant for the developer's daily work. Consequently, naming of variables, parameters, classes, methods, function modules etc. will play a major role in this document.

The name of an ABAP object or ABAP programming element should reflect

- ✓ To which module belongs the element
- ✓ In which area is the element located
- ✓ If possible, what kind of programming element it is.

3.3 General Syntax Elements

COMPONENT	DESCRIPTION
<>	Every element that is variable is displayed in spear brackets.
[ ]	Every element that is optional is displayed in square brackets.
<name>	You can use a character string of your choice for the symbol **<name>**. Make sure that this string is object related, so that the name of the object will be meaningful and understandable. Use English language for this string. Underscores must be used to separate parts of names. If there is already an object with the equal name you can use sequential numbering.
<component>	Component = Module information (obligatory): □ BC Basis module (common programs) □ CA Cross functions module □ FI FI module □ MM MM module □ SD SD module □ CO CO module

	□ PP PP module □ PM PM module □ QM QM module
<activity>	The activity is by grammar a verb that describes a certain activity in order to manipulate an object or provide a service for the object. The activity should be as descriptive as possible. Abbreviations should be avoided if possible. If an activity is clearly defined, the same naming should be done for it wherever needed. In order to add more detail to the activity, variants (see below) can be added to it. An activity never represents a name. An example list of standardized activities is: NEW/CREATE create a new object instance GET read buffered from memory SET replace memory with new data ADD add one entry (in memory) CHANGE changes certain values (in memory) REMOVE remove one entry (in memory) READ read from database (unbuffered) WRITE write to database INSERT insert into database DELETE delete from database UPDATE changes certain values on database CHECK check routine returning a descriptive message IS/DOES/HAS request for flag status (Boolean information) INIT[IALIZE] initial setup of program/memory RESET clear memory (of program)

3.4 General Rule

The general naming rule for objects not specially mentioned in this document is Z**<name>.**

Template: *ZPRODUCTION_LIST_ORDERSZSALES_REPORTING*
ZCA_FORMS

3.5 Naming Conventions for Packages

- ✓ Custom developed objects (programs, search helps, function modules, etc.) should be subdivided into logical units using different development classes/packages. The development packages will be unique.
- ✓ When saving a new piece of development (program, fdata dictionary object etc.), a development package has to be assigned.
- ✓ If this new object is only for testing purposes and never supposed to be transported into another environment, create it as a ‘local private object’ (development package $TMP). In this way no ‘unnecessary’ change requests will be created.
- ✓ If the new object is supposed to be transported to other environments, assign it to the appropriate development package.

Packages are subject to the following naming conventions.
General rules: **Z<name>**

Examples:
Sub Packages: ZPRODUCTION_LIST_ORDERS
ZSALES_REPORTING
ZCA_FORMS
ZEVENT_MANAGEMENT

3.6 Naming Conventions for DDIC Objects

OBJECT NAMING	CONVENTION	EXAMPLE
domains	ZD_<name>	ZD_EXAMPLE
data elements	ZE_<name>	ZE_EXAMPLE
table types	Z<component>_TT_<name>	ZCO_TT_EXAMPLE
structures	Z<component>_S_<name>	ZCO_S_EXAMPLE
views	Z<component>_V_<name>	ZCO_V_EXAMPLE
view clusters	Z<component>_VC_<name>	ZCO_VC_EXAMPLE
database tables	Z<component>_T_<name>	ZCO_T_EXAMPLE
text tables	<name of corresponding table>_T	ZCO_T_EXAMPLE_T

3.7 Naming Convention for ABAP Objects

3.7 ABAP Main Objects

3.7 OBJECT	3.7 SYNTAX	3.7 EXAMPLE
report	Z<component>_<name>	ZCO_READ_COEP
TOP include	<program name>_TOP	ZCO_READ_COEP_TOP
other includes	<program name>_<variant>	ZCO_READ_COEP_BYDATE
module pool	SAPMZ<component>_<name>	SAPMZSD_REBATE
module pool top include	MZ<module name>_TOP	MZSD_REBATE_TOP
module pool PBO include	MZ<module name>_O<number>	MZSD_REBATE_O01
module pool PAI include	MZ<module name>_I<number>	MZSD_REBATE_I01
module pool form include MZ	M<module name>_F<number>	MZSD_REBATE_I01
module pool screens SAPMZ	SAPMZ<component>_<screen number>	SAPMZSD_REBATE_0110
function group	Z<component>_<name>	ZCO_PLAN
function group top include	L<function group name>_TOP	LZCO_PLAN_TOP
function group form include	L<function group name>_F<number>	LZCO_PLAN_F01

function modules	Z<component>_<object>_<activity>	ZCO_INVOICE_READ
class(*)	ZCL_<component>_<name>	ZCL_CO_EXAMPLE
class (BADI impl.)	ZCL_IM_<component>_<name>	ZCL_IM_CA_EXAMPLE
exception class(*)	ZCX_<component>_<name>	ZCX_CO_EXAMPLE
interface(*)	ZIF_<component>_<name>	ZIF_CO_EXAMPLE
constant-interface	ZIF_<component>_<name>_C	ZIF_CO_EXAMPLE_C
transaction	Z<component>_<name>	ZCO_EXAMPLE
message class	ZME_<component>_<name>	ZME_CO_EXAMPLE

(*)Local classes/interfaces should start with the prefix LCL_.

3.8 Methods

OBJECT	SYNTAX	EXAMPLE
class methods	<activity>_[<object>_]	SEND_RECEIPT
events	<occurred event>(**)	INVOICE_FILTERED
event handler	ON_<occurred event>	ON_INVOICE_FILTERED

(**) The naming of the event should be in present perfect like in the example above.

3.8 Parameters

When choosing method parameters, the following should be considered:

- Better to split methods than add OPTIONAL parameters
- Split method instead of using boolean input parameters
- Use a small number of IMPORTING parameters, ideally less than three
- RETURN, EXPORT or CHANGE - and only exactly one parameter
- Better RETURNING than EXPORTING
- RETURNING from large tables is usually okay
- Use CHANGING sparingly

3.8 Method Declaration Type

A method should usually be declared as an instance method. Static methods should only be used if there is a justified need.

3.8 Visibility

Methods should be declared private and only changed to protected or public when there is a legitimate need
Only selected methods may be called from outside. These receive the visibility public.

3.8 Call Method

ABAP Objects methods should be written functionally (i.e. without CALL METHOD). To do this, the functional Notation for CALL METHOD function can be activated in the SE80 ABAP Editor under the Pattern tab.

Example:
lf_name = lo_customer->get_name( ).

3.9 Memory Elements

Memory elements are subject to the following naming conventions:
<visibility prefix><data type>_<name>

VISIBILITY ABAP		DATA TYPE (<DATA TYPE>)	
L	Local	C	Constant
G	Global	V	Variable
S	Static	S	Structure
VISIBILITY WITHIN ABAP OO		T	Table (or table type)
G	static attribute (class attribute)	R	Data reference
M	instance (member) attribute	O	Object reference
G	constants (class attribute)	B	BADI reference
		X	Exception reference
		P	Parameter
		SO	Select option
		RA	Ranges

Examples:

tables:

- GT_VENDORS — global table
- LT_VENDORS — local table

structures:

- GS_VENDOR — global structure
- LS_VENDOR — local structure

fields:

- GV_CATEGORY — global variable
- LV_LINES — local variable

constants:

- GC_DISPLAY — global constant / class constants
- LC_DISPLAY — local constant

objects, instances:

- GB_BADI — global BADI reference
- LO_OBJECT — local reference
- LX_FAIL — local exception

3.10 Signature Elements

Signature elements are subject to the following naming conventions
<visibility prefix><data type>_<name>

VISIBILITY ABAP	DATA TYPE (<DATA TYPE>)

I	Importing parameter	C	Constant
E	Exporting parameter	V	Variable
C	Changing parameter	S	Structure
R	Returning parameter	T	Table (or table type)
I	Using parameter	R	Data reference
		O	Object reference
		B	Badi reference
		X	Exception reference
		P	Parameter
		SO	Select option
		RA	Ranges

Examples:

Tables:

- IT_VENDORS — subroutine/function module import table
- ET_VENDORS — subroutine/function module export table
- CT_VENDORS — subroutine import and export table

Structures:

- IS_VENDOR Fields: — function module import structure

- IV_CATEGORY — subroutine/method import parameters
- EV_CATEGORY — subroutine/method export parameters

3.11 Other Constructs

OBJECT	SYNTAX	EXAMPLE
types	<visibility pre-fix>TY_<name>	LTY_MY_LOCAL_TYPE GTY_MY_GLOBAL_TYPES
field symbols(*)	<<visibility prefix><data type>_<name>>	<LS_EXAMPLE>
search helps (elementary)	Z<component>_ELM_<name>	ZCO_ELM_PROCESSED_ITEMS
search helps (compound)	Z<component>_CMP_<name>	ZCO_CMP_MISSING_ITEMS

(*) Field symbols that are declared as TYPE ANY are named with <data type> G (generic).

3.12 Enhancement Implementations

COMPONENT	DESCRIPTION
<process>	The process describes the function
<sourceprogram>	The soruceprogram is the name of the program where the enhancement is implemented
<sourcecodepoint>	The sorucecodepoint is the name of the program where the enhancement is implemented

<sub_process>	The sub_process is a logical part of the process
<further_break_down_of_the_sub_process>	The futher_break_down_of_the_sub_process is a logical part of the sub process

Description	name	package	top-package	example
Explicit and implicit enhancment (Source-Plugin)				
Composite enhancement spots	ZZES_<process>	Z<component>_SPOTS	ZCUSTOMER_FUNCTION	
Enhancement spots (Source-Plugin)	ZES_<sourceprogram>	Z<component>_SPOTS	ZCUSTOMER_FUNCTION	ENHANCEMENT-SECTION ZESEC_xxx SPOTS **ZES_BADI_PO.**
Enhancement option Enhance. impl. **point**	ZEPOI_<sourcecodepoint>	Z<component>_SPOTS	ZCUSTOMER_FUNCTION	ENHANCEMENT-POINT **ZEPOI_xxx** SPOTS ZES_BADI_PO.
Enhancement option enhance. impl. **Section**	ZESEC_<sourcecodepoint>	Z<component>_SPOTS	ZCUSTOMER_FUNCTION	ENHANCEMENT-Section **ZESEC_xxx** SPOTS ZES_BADI_PO.
explicit or implicit enhancement mplementation	Z500_<process>	Z<process>	ZPROCESSES	ENHANCEMENT 1 **Z500_INSTRUCTION_MANUAL_MM.**
Explicit BADI-enhancement (Object-Plugin)				
Enhancement spot (Object-Plugin)	ZBES_<process>	Z<component>_SPOTS	ZCUSTOMER_FUNCTION	
BADI-Definition	ZBADI_<process>	Zcomponent>_SPOTS	ZCUSTOMER_FUNCTION	
Interface	ZBIF_<process>	Z<component>_SPOTS	ZCUSTOMER_FUNCTION	
enhancement implementation	ZBEI_<sub_process>	Z<process>	ZPROCESSES	
enhancement Impl. element (BADI-implementation)	ZBEIE_<further_break_down_of_the_sub_process>	Z<process> or Z<sub_process>	ZPROCESSES	Package and top package depending on the desired switchability of the implementation
implementing class	ZBCL_<further_break_down_of_the_sub_process>	Z<process> or Z<sub_process>	ZPROCESSES	Package and top package depending on the desired switchability of the implementation
Expliziter BADI-Sorter				
enhancement Implementation	ZBEI_SORTER_<process>	Z<process>	ZPROCESSES	

enhancement Impl. element (BADI-Implementation)	ZBEIE_SORTER_<process>	Z<process>	ZPROCESSES	
implementing class	ZBCL_SORTER_<process>	Z<process>	ZPROCESSES	If you only want to sort by order, ZBCL_SORTER_SEQUENCE can also be used
Subscreen function group	ZBFG_SORTER_<process>	Z<process>	ZPROCESSES	If you only want to sort by order, ZBCL_SORTER_SEQUENCE can also be used
Switch Framework				
Switch	ZSW_<process>	ZSwitch_Framework	ZCUSTOMER_FUNCTION	
Business Function Set	ZBFS_<process>	ZSwitch_Framework	ZCUSTOMER_FUNCTION	
Business Function	ZBF_<process>	ZSwitch_Framework	ZCUSTOMER_FUNCTION	

3.13 ABAP CDS Views

The picture below shows an example of the numbering of the elements used and defined in this chapter.

```
[C80] Z_I_SALESORDER_FOR_ORDERTYPE   1
1  @AbapCatalog.sqlViewName:  ZI_SO_FOR_ORDTPE   2
2  @AbapCatalog.compiler.compareFilter: true
3  @AbapCatalog.preserveKey: true
4  @AccessControl.authorizationCheck: #CHECK
5  @EndUserText.label: 'Salesorder for requested ordertypes'
6  define view Z_I_SalesOrder_for_OrderType   3
7    with parameters
8      p_auart : auart
9    as select from vbak as k
10     inner join   vbap as p on k.vbeln = p.vbeln
11 {
12   key k.vbeln as SalesDocument,
13   key p.posnr as SalesDocumentItem,
14       k.vkorg as SalesOrganization,
15       k.auart as SalesDocumentType,
16       p.matnr as Material,
17       p.zmeng as TargetQuantitiy,
18       p.zieme as TargetQuantityUnit
19 }
20 where
21   k.auart = :p_auart
```

View	View-Type	Name data definition (1)	package @AbapCatalog.sqlViewName (2)	Name View (3)
BasicInterface Views	@VDM.viewType: #BASIC	Z_I_*	ZI_*	Z_I_*
Composite-Interface-View	@VDM.viewType: #COMPOSITE	Z_I_*	ZI_*	Z_I_*
Consumption Views	@VDM.viewType: #CONSUMPTION	Z_C_*	ZC_*	Z_C_*
Transaktionaler Objek-View	@VDM.viewType: #TRANSACTIONAL	—	—	—
Restricted-Reuse-View	@VDM.lifecycle.contract.type: #SAP_INTERNAL_API	—	—	—
Transaktionaler Consumption-View	@VDM.viewType: #CONSUMPTION @ObjectModel. transactionalProcessing-Delegated:true	—	—	—
Remote API View	@VDM.lifecycle.contract.type: #PUBLIC_REMOTE_API	Z_A_*	ZA_*	Z_A_*
Privater VDM_View	@VDM.private: true	—	—	—
Extension-Include-View	@VDM.viewType: #EXTENSION	Z_E_*	ZE_*	Z_E_*
VDM-View-Erweiterung	@VDM.viewExtension: true	Z_X_*	ZX_*	Z_X_*

The most important CDS types:

- **Z_I_** -> for public CDS types: view (I=Interface)
  - Direct use in custom views
  - Example: Z_I_SalesArea
- ***QRY or *Q**
  - Ready to use analytic Query, Odata Service ready
  - Example: Z_I_SalesOrderItemProcessQ
- **Z_C_** -> for Consumption views
  - Beispiel: Z_C_SalesOrderItemProcessQ
- **Z_E_** -> for CDS view extensions
  - Example: Z_E_I_SALESORDER (Erweiterung des CDS Views I_SALESORDER)
- ***ValueHelp**
  - Example: Z_I_Sales_Office_ValueHelp

With regard to the column description (apart from descriptive names), no further specifications are made. Using Camel Case is standard in this environment.

3.14 ABAP RESTful Application Programming Model

For important information, see SAP - ABAP RESTful Application Programming Model.

OBJECT	SYNTAX	EXAMPLE
Service Definition	Z_<name>	Z_SALES_ORDER
Service Binding	• Z_<API/UI>_ <Bezeichnung>_ <ODATA Version> • <API/UI>: UI für UI-Services, API für Web-API • <ODATA Version>: O2 oder O4	Z_API_SALES_ORDER_O2
Behavior Pool/ Behavoior Class	ZBP_<CDS_NAME >	ZBP_I_SALES_ORDER
Local Handler Class	lhc_<CDS_NAME >	lhc_Z_I_Sales
Local Saver Class	lsc_<CDS_NAME >	lsc_Z_I_Sales

3.15 Naming Convention for Transport Request

CONVENTION	EXAMPLE
<Number Requirement_<module_name>_<name>_<date YYYY-MM-DD>	S8000003525_SD_price determination book_2021-03-30

4 Development Guidelines

4.1 General

The sense of following guidelines is to have a stable and maintainable system and it offers the possibility to split local and global changes to minimize the impact on the system.

4.1 Texts

Text should be created as text variables so these can be translated using the Standard SAP tool for other languages.

4.1 Obsolete statements

Please do not use statements that are marked as obsolete from SAP.

4.1 Field-Symbols / References

If possible always use field-symbols or references instead of work areas because of better performance.

4.1 Comments

Comments have to be written in English and should not comment on obvious effects of ABAP statements. They should always describe why code does what it does and not how it achieves it. Please do not use comments for tracking which changes were done when and by whom because for change tracking you can use the version history of SAP.

4.1 Pretty Printer

The pretty printer should always be used with the following settings:

- ✓ Indent statements
- ✓ Do not insert standard comment
- ✓ Keywords uppercase

4.1 Updates

Use a BAPI if it available. If not available, use a SAP function module.
When both not available, capsulate the “update” statement in a function module and use the SAP lock logic.
Updates have to run “in Update Task”.

4.1 Filesystem

If you would like to write files in background to any kind of share you have to define the filename via transaction FILE. It is not allowed any more to write directly to the filesystem of the SAP-Server, Instead you should use the shared fileserver.

To transfer your file to the shared fileserver you must use the variable <V=ZFILESHARE>, as you see in the example below.

This variable points to the directory: //sapfiles.stiebel-eltron.com/sap/<sysid>_<client>

Under this directory you can create subfolders which has to be named speaking.

Example:

Logical path	ZSD_EXPORT_INVOICES	
Name		
Syntax group	WINDOWS NT	Microsoft Windows NT
Physical path	<V=ZFILESHARE>\ExportInvoices\<FILENAME>	

4.2 Copying Sap Standard Objects / Global Enhancements

Copying of SAP Standard objects (e.g. reports, function modules) is not allowed, because upgrades wouldn’t affect them and therefore coding always stays in the copied old version. Enhancements can be used instead.

4.3 User-exits, BADI's, enhancement's

If you want to modify the SAP standard process, check the following options in the order:

1. Check explicit enhancement option or BADI is available
2. Check if an user-exit is available
3. Check implicit enhancement option is available
4. Modification on SAP standard source codes should be avoided. If it is necessary, the head of application development must approve it. Create explicit or implicit enhancement in it and implement them.

4.3 No functional coding in the user exit

All functions should be implemented in a separate encapsulated module (OO method, very rarely function module) and integrated in the exit by calling this module.

4.3 Userexit control

The user exit control is available in the package /ITS/DI_2_BC_EXIT_CONTROL.
Individual functions can be equalized by implementing the user exit control and made switchable via a control table.

If necessary, the check logic can be extended by enhancements in the exit control. In addition, SAP standard GUIs can be expanded with additional functions without modification

4.4 Forms

Adobe forms should be used for global and local development requests.

Often forms are copied from existing SAP Standard forms. If a form is copied retain the existing name as a reference with a version.
Example: MEDRUCK_PO (SAP Standard form)
ZMM_MEDRUCK (Copied form name)

4.5 Database Selection

Use SELECT SINGLE whenever possible, since this is the fastest way to retrieve data from databases. Also, if there is more than one key field, list the key fields in the order that they are in the table definition. This will also improve system performance.

If you are hitting the databases twice (or even more often) from the same record, you should "buffer" the required information in an internal table and check this table before reading the database again. In this instance, use SELECT … INTO TABLE TBL_XXXXX as this is much more efficient than a SELECT … ENDSELECT loop using the APPEND statement.

Do not nest SELECT statements, i.e. have SELECT loops within another SELECT loop. Instead define a VIEW in the data dictionary (if the volume of data is large), use the SELECT … FOR ALL ENTRIES, or use the SELECT … INTO TABLE statement.

In a SELECT always specify all known conditions in the WHERE clause (if possible). If there is no WHERE clause the DBMS has no chance to make optimizations. Do not leave the WHERE clause out and use the CHECK statement in the select loop (as this is very inefficient).

Try and keep the amount of data transferred from the database to the application small. If only certain fields are required from a table, use the SELECT field1 field2 INTO (field1, field2) FROM TABLE statement.

4.6 Authorization checks

In general, as part of the technical requirements definition, it is to be determined how detailed a Authorization check should take place:

- No
- Basic - transaction level checking
- Extended - check on organization level such as sales organization, plant, company code or purchasing organization
- Maximum - field level check

The “basic authorization check” only makes sense if all programs can be called using their own transactions.
An individual parameter transaction must be created for generated table maintenance views.

4.7 Time zones

Date and time have to be saved as timestamps (time zone UTC) in customer tables. Use ‘GET TIME STAMP FIELD’ for getting current UTC timestamps.

5 Appendix

APPENDIX		
Document name	**Title**	**Link**